SANDIA REPORT

SAND2014-18874
Unlimited Release
Printed October 2014

MueLu User's Guide 1.0 (Trilinos version 11.12)

Andrey Prokopenko, Jonathan J. Hu, Tobias A. Wiesner, Christopher M. Siefert, Raymond S. Tuminaro

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-18874
Unlimited Release
Printed October 2014

MueLu User's Guide 1.0
(Trilinos version 11.12)

Andrey Prokopenko
Scalable Algorithms
Sandia National Laboratories
Mailstop 1318
P.O. Box 5800
Albuquerque, NM 87185-1318
aprokop@sandia.gov

Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany
wiesner@lnm.mw.tum.de

Jonathan J. Hu
Scalable Algorithms
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
jhu@sandia.gov

Christopher M. Siefert
Computational Multiphysics
Sandia National Laboratories
Mailstop 1322
P.O. Box 5800
Albuquerque, NM 87185-1322
csiefer@sandia.gov

Raymond S. Tuminaro
Computational Mathematics
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
rstumin@sandia.gov

Abstract

This is the official user guide for MUELU multigrid library in Trilinos version 11.12. This guide provides an overview of MUELU, its capabilities, and instructions for new users who want to start using MUELU with a minimum of effort. Detailed information is given on how to drive MUELU through its XML interface. Links to more advanced use cases are given. This guide gives information on how to achieve good parallel performance, as well as how to introduce new algorithms. Finally, readers will find a comprehensive listing of available MUELU options. *Any options not documented in this manual should be considered strictly experimental.*

Acknowledgment

Many people have helped develop MUELU and/or provided valuable feedback, and we would like to acknowledge their contributions here: Tom Benson, Julian Cortial, Eric Cyr, Stefan Domino, Travis Fisher, Jeremie Gaidamour, Axel Gerstenberger, Chetan Jhurani, Mark Hoemmen, Paul Lin, Eric Phipps, Siva Rajamanickam, Nico Schlömer, and Paul Tsuji.

Contents

1 Introduction **11**

2 Multigrid background **13**

3 Getting Started **15**

3.1 Overview of MueLu 15

3.2 Configuration and Build 16

3.2.1 Dependencies 16

3.2.2 Configuration 18

3.3 Examples in code 18

3.3.1 MueLu as a preconditioner within Belos 19

3.3.2 MueLu as a preconditioner for AztecOO 21

3.3.3 Further remarks 22

4 Performance tips **23**

5 MueLu options **25**

5.1 Using parameters on individual levels 25

5.2 Parameter validation 25

5.3 General options 26

5.4 Smoothing and coarse solver options 27

5.5 Aggregation options 29

5.6 Rebalancing options 30

5.7 Multigrid algorithm options 31

5.8 Miscellaneous options 33

References **34**

Appendix

A Copyright and License **37**

List of Figures

List of Tables

3.1 MUELU's required and optional dependencies, subdivided by whether a dependency is that of the MUELU library itself (*Library*), or of some MUELU test (*Testing*). 17

5.1 Supported problem types ("–" means not set). 26

5.2 Commonly used smoothers provided by IFPACK/IFPACK2. Note that these smoothers can also be used as coarse grid solvers. 28

5.3 Commonly used direct solvers provided by AMESOS/AMESOS2 28

5.4 Available coarsening schemes. 30

5.5 Available multigrid algorithms for generating grid transfer matrices. 32

Chapter 1

Introduction

This guide gives an overview of MUELU's capabilities. If you are looking for a tutorial, please refer to the MUELU tutorial in `muelu/doc/Tutorial` (see also [17]). New users should start with §3. It strives to give the new user all the information he/she might need to begin using MUELU quickly. Users interested in performance, especially in parallel context, should refer to §4. Users looking for a particular option should consult §5, containing a complete set of supported options in MUELU.

If you find any errors or omissions in this guide, have comments or suggestions, or would like to contribute to MUELU development, please contact the MUELU users list, or developers list.

Chapter 2

Multigrid background

Here we provide a brief multigrid introduction (see [5] or [14] for more information). A multigrid solver tries to approximate the original problem of interest with a sequence of smaller (*coarser*) problems. The solutions from the coarser problems are combined in order to accelerate convergence of the original (*fine*) problem on the finest grid. A simple multilevel iteration is illustrated in Algorithm 1.

Algorithm 1 V-cycle multigrid with N levels to solve $Ax = b$.

$A_0 = A$
function MULTILEVEL(A_k, b, u, k)
  // Solve A_k u = b (k is current grid level)
  $u = S_m^1(A_k, b, u)$
  if $(k \neq \mathbf{N} - \mathbf{1})$ **then**
    P_k = determine_interpolant(A_k)
    R_k = determine_restrictor(A_k)
    $\widehat{r}_{k+1} = R_k(b - A_k u)$
    $A_{k+1} = R_k A_k P_k$
    $v = 0$
    MULTILEVEL($\widehat{A}_{k+1}$, $\widehat{r}_{k+1}$, v, $k+1$)
    $u = u + P_k v$
    $u = S_m^2(A_k, b, u)$
  end if
end function

In the multigrid iteration in Algorithm 1, the $S_m^1()$'s and $S_m^2()$'s are called *pre-smoothers* and *post-smoothers*. They are approximate solvers (e.g., symmetric Gauss-Seidel), with the subscript m denoting the number of applications of the approximate solution method. The purpose of a smoother is to quickly reduce certain error modes in the approximate solution on a level k. For symmetric problems, the pre- and post-smoothers should be chosen to maintain symmetry (e.g., forward Gauss-Seidel for the pre-smoother and backward Gauss-Seidel for the post-smoother). The P_k's are *interpolation* matrices that transfer solutions from coarse levels to finer levels. The R_k's are *restriction* matrices that restrict a fine level solution to a coarser level. In a geometric multigrid, P_k's and R_k's are determined by the application, whereas in an algebraic multigrid they are automatically generated. For symmetric problems, typically $R_k = P_k^T$. For nonsymmetric problems, this is not necessarily true. The A_k's are the coarse level problems, and are generated through

a Galerkin (triple matrix) product.

Please note that the algebraic multigrid algorithms implemented in MUELU generate the grid transfers P_k automatically and the coarse problems A_k via a sparse triple matrix product. TRILINOS provides a wide selection of smoothers and direct solvers for use in MUELU through the IFPACK, IFPACK2, AMESOS, and AMESOS2 packages (see §5).

Chapter 3

Getting Started

This section is meant to get you using MUELU as quickly as possible. §3 gives a summary of MUELU's design. §3.2 lists MUELU's dependencies on other TRILINOS libraries and provides a sample cmake configuration line. Finally, code examples using the XML interface are given in §3.3.

3.1 Overview of MUELU

MUELU is an extensible algebraic multigrid (AMG) library that is part of the TRILINOS project. MUELU works with EPETRA (32-bit version [1]) and TPETRA matrix types. The library is written in C++ and allows for different ordinal (index) and scalar types. MUELU is designed to be efficient on many different computer architectures, from workstations to supercomputers, relying on "MPI+X" principle, where "X" can be threading or CUDA.

MUELU provides a number of different multigrid algorithms:

1. smoothed aggregation AMG (for Poisson-like and elasticity problems);
2. Petrov-Galerkin aggregation AMG (for convection-diffusion problems);
3. energy-minimizing AMG;
4. aggregation-based AMG for problems arising from the eddy current formulation of Maxwell's equations.

MUELU's software design allows for the rapid introduction of new multigrid algorithms. The most important features of MUELU can be summarized as:

Easy-to-use interface

MUELU has a user-friendly parameter input deck which covers most important use cases. Reasonable defaults are provided for common problem types (see Table 5.1).

[1] Support for the Epetra 64-bit version is planned.

Modern object-oriented software architecture

MUELU is written completely in C++ as a modular object-oriented multigrid framework, which provides flexibility to combine and reuse existing components to develop novel multigrid methods.

Extensibility

Due to its flexible design, MUELU is an excellent toolkit for research on novel multigrid concepts. Experienced multigrid users have full access to the underlying framework through an advanced XML based interface. Expert users may use and extend the C++ API directly.

Integration with TRILINOS library

As a package of TRILINOS, MUELU is well integrated into the TRILINOS environment. MUELU can be used with either the TPETRA or EPETRA (32-bit) linear algebra stack. It is templated on the local index, global index, scalar, and compute node types. This makes MUELU ready for future developments.

Broad range of supported platforms

MUELU runs on wide variety of architectures, from desktop workstations to parallel Linux clusters and supercomputers ([10]).

Open source

MUELU is freely available through a simplified BSD license (see Appendix A).

3.2 Configuration and Build

MUELU has been compiled successfully under Linux with the following C++ compilers: GNU versions 4.1 and later, Intel versions 12.1/13.1, and clang versions 3.2 and later. In the future, we recommend using compilers supporting C++11 standard.

3.2.1 Dependencies

Required Dependencies

MUELU requires that TEUCHOS and either EPETRA/IFPACK or TPETRA/IFPACK2 are enabled.

Recommended Dependencies

We strongly recommend that you enable the following TRILINOS libraries along with MUELU:

- EPETRA stack: AZTECOO, EPETRA, AMESOS, IFPACK, ISORROPIA, GALERI, ZOLTAN;

- TPETRA stack: AMESOS2, BELOS, GALERI, IFPACK2, TPETRA, ZOLTAN2.

Tutorial Dependencies

In order to run the MUELU Tutorial [17] located in `muelu/doc/Tutorial`, MUELU must be configured with the following dependencies enabled:

AZTECOO, AMESOS, AMESOS2, BELOS, EPETRA, IFPACK, IFPACK2, ISORROPIA, GALERI, TPETRA, ZOLTAN, ZOLTAN2.

☛ Note that the MUELU tutorial [17] comes with a VirtualBox image with a pre-installed Linux and TRILINOS. In this way, a user can immediately begin experimenting with MUELU without having to install the TRILINOS libraries. Therefore, it is an ideal starting point to get in touch with MUELU.

Complete List of Direct Dependencies

Dependency	Required		Optional	
	Library	Testing	Library	Testing
AMESOS			×	×
AMESOS2			×	×
AZTECOO				×
BELOS				×
EPETRA			×	×
IFPACK			×	×
IFPACK2			×	×
ISORROPIA			×	×
GALERI				×
KOKKOSCLASSIC			×	
TEUCHOS	×	×		
TPETRA			×	×
XPETRA	×	×		
ZOLTAN			×	×
ZOLTAN2			×	×
Boost			×	
BLAS	×	×		
LAPACK	×	×		
MPI			×	×

Table 3.1. MUELU's required and optional dependencies, subdivided by whether a dependency is that of the MUELU library itself (*Library*), or of some MUELU test (*Testing*).

Table 3.1 lists the dependencies of MUELU, both required and optional. If an optional dependency is not present, the tests requiring that dependency are not built.

☛ AMESOS/AMESOS2 are necessary if one wants to use a sparse direct solve on the coarsest level. ZOLTAN/ZOLTAN2 are necessary if one wants to use matrix rebalancing in parallel runs (see §4). AZTECOO/BELOS are necessary if one wants to test MUELU as a preconditioner instead of a solver.

☛ MUELU has also been successfully tested with SuperLU 4.1 and SuperLU 4.2.

☛ Some packages that MUELU depends on may come with additional requirements for third party libraries, which are not listed here as explicit dependencies of MUELU. It is highly recommended to install ParMetis 3.1.1 or newer for ZOLTAN, ParMetis 4.0.x for ZOLTAN2, and SuperLU 4.1 or newer for AMESOS/AMESOS2.

3.2.2 Configuration

The preferred way to configure and build MUELU is to do that outside of the source directory. Here we provide a sample configure script that will enable MUELU and all of its optional dependencies:

```
export TRILINOS_HOME=/path/to/your/Trilinos/source/directory
cmake -D BUILD_SHARED_LIBS:BOOL=ON \
      -D CMAKE_BUILD_TYPE:STRING="RELEASE" \
      -D CMAKE_CXX_FLAGS:STRING="-g" \
      -D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
      -D Trilinos_ENABLE_TESTS:BOOL=OFF \
      -D Trilinos_ENABLE_EXAMPLES:BOOL=OFF \
      -D Trilinos_ENABLE_MueLu:BOOL=ON \
      -D MueLu_ENABLE_TESTS:STRING=ON \
      -D MueLu_ENABLE_EXAMPLES:STRING=ON \
      -D TPL_ENABLE_BLAS:BOOL=ON \
      -D TPL_ENABLE_MPI:BOOL=ON \
      ${TRILINOS_HOME}
```

More configure examples can be found in `Trilinos/sampleScripts`. For more information on configuring, see the TRILINOS Cmake Quickstart guide [1].

3.3 Examples in code

The most commonly used scenario involving MUELU is using a multigrid preconditioner inside an iterative linear solver. In TRILINOS, a user has a choice between EPETRA and TPETRA

for the underlying linear algebra library. Important Krylov subspace methods (such as preconditioned CG and GMRES) are provided in TRILINOS packages AZTECOO (EPETRA) and BELOS (EPETRA/TPETRA).

At this point, we assume that the reader is comfortable with TEUCHOS referenced-counted pointers (RCPs) for memory management (an introduction to RCPs can be found in [3]) and the `Teuchos::ParameterList` class [13].

3.3.1 MUELU as a preconditioner within BELOS

The following code shows the basic steps of how to use a MUELU multigrid preconditioner with TPETRA linear algebra library and with a linear solver from BELOS. To keep the example short and clear, we skip the template parameters and focus on the algorithmic outline of setting up a linear solver. For further details, a user may refer to the `examples` and `test` directories.

First, we create the MUELU multigrid preconditioner. It can be done in a few ways. For instance, multigrid parameters can be read from an XML file (e.g., *mueluOptions.xml* in the example below).

```
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...
std::string optionsFile = "mueluOptions.xml";
Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner =
   MueLu::CreateTpetraPreconditioner(A, optionsFile);
```

The XML file contains multigrid options. A typical file with MUELU parameters looks like the following.

```
<ParameterList name="MueLu">

  <Parameter name="verbosity" type="string" value="low"/>

  <Parameter name="max levels" type="int" value="3"/>
  <Parameter name="coarse: max size" type="int" value="10"/>

  <Parameter name="multigrid algorithm" type="string" value="sa"/>

  <!-- Damped Jacobi smoothing -->
  <Parameter name="smoother: type" type="string" value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type" type="string" value="Jacobi"/>
    <Parameter name="relaxation: sweeps" type="int" value="1"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.9"/>
  </ParameterList>

  <!-- Aggregation -->
```

```
  <Parameter name="aggregation: type" type="string" value="uncoupled"/>
  <Parameter name="aggregation: min agg size" type="int" value="3"/>
  <Parameter name="aggregation: max agg size" type="int" value="9"/>

</ParameterList>
```

It defines a three level smoothed aggregation multigrid algorithm. The aggregation size is between three and nine(2D)/27(3D) nodes. One sweep with a damped Jacobi method is used as a level smoother. By default, a direct solver is applied on the coarsest level. A complete list of available parameters and valid parameter choices is given in §5 of this User's Guide.

Users can also construct a multigrid preconditioner using a provided `ParameterList` without accessing any files in the following manner.

```
  Teuchos::RCP<Tpetra::CrsMatrix<> > A;
  // create A here ...
  Teuchos::ParameterList paramList;
  paramList.set("verbosity", "low");
  paramList.set("max levels", 3);
  paramList.set("coarse: max size", 10);
  paramList.set("multigrid algorithm", "sa");
  // ...
  Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner =
     MueLu::CreateTpetraPreconditioner(A, paramList);
```

Besides the linear operator A, we also need an initial guess vector for the solution X and a right hand side vector B for solving a linear system.

```
    Teuchos::RCP<const Tpetra::Map<> > map = A->getDomainMap();

    // Create initial vectors
    Teuchos::RCP<Tpetra::MultiVector<> > B, X;
    X = Teuchos::rcp( new Tpetra::MultiVector<>(map,numrhs) );
    Belos::MultiVecTraits<>::MvRandom( *X );
    B = Teuchos::rcp( new Tpetra::MultiVector<>(map,numrhs) );
    Belos::OperatorTraits<>::Apply( *A, *X, *B );
    Belos::MultiVecTraits<>::MvInit( *X, 0.0 );
```

To generate a dummy example, the above code first declares two vectors. Then, a right hand side vector is calculated as the matrix-vector product of a random vector with the operator A. Finally, an initial guess is initialized with zeros.

Then, one can define a `Belos::LinearProblem` object where the `mueLuPreconditioner` is used for left preconditioning

```
    Belos::LinearProblem<> problem( A, X, B );
    problem->setLeftPrec(mueLuPreconditioner);
```

```
bool set = problem.setProblem();
```

Next, we set up a BELOS solver using some basic parameters

```
Teuchos::ParameterList belosList;
belosList.set( "Block Size", 1 );
belosList.set( "Maximum Iterations", 100 );
belosList.set( "Convergence Tolerance", 1e-10 );
belosList.set( "Output Frequency", 1 );
belosList.set( "Verbosity", Belos::TimingDetails + Belos::FinalSummary );

Belos::BlockCGSolMgr<> solver( rcp(&problem,false), rcp(&belosList,false) );
```

Finally, we solve the system.

```
Belos::ReturnType ret = solver.solve();
```

3.3.2 MUELU as a preconditioner for AZTECOO

For EPETRA, users have two library options: BELOS (recommended) and AZTECOO. AZTECOO and BELOS both provide fast and mature implementations of common iterative Krylov linear solvers. BELOS has additional capabilities, such as Krylov subspace recycling and "tall skinny QR".

Constructing a MUELU preconditioner for Epetra operators is done in a similar manner to Tpetra.

```
Teuchos::RCP<Epetra_CrsMatrix> A;
// create A here ...
Teuchos::RCP<MueLu::EpetraOperator> mueLuPreconditioner;
std::string optionsFile = "mueluOptions.xml";
mueLuPreconditioner = MueLu::CreateEpetraPreconditioner(A, optionsFile);
```

MUELU parameters are generally Epetra/Tpetra agnostic, thus the XML parameter file could be the same as §3.3.1.

Furthermore, we assume that a right hand side vector and a solution vector with the initial guess are defined.

```
Teuchos::RCP<const Epetra_Map> map = A->DomainMap();
Teuchos::RCP<Epetra_Vector> B = Teuchos::rcp(new Epetra_Vector(map));
Teuchos::RCP<Epetra_Vector> X = Teuchos::rcp(new Epetra_Vector(map));
X->PutScalar(0.0);
```

Then, an `Epetra_LinearProblem` can be defined.

```
Epetra_LinearProblem epetraProblem(A.get(), X.get(), B.get());
```

The following code constructs an AZTECOO CG solver.

```
AztecOO aztecSolver(epetraProblem);
aztecSolver.SetAztecOption(AZ_solver, AZ_cg);
aztecSolver.SetPrecOperator(mueLuPreconditioner.get());
```

Finally, the linear system is solved.

```
int maxIts = 100;
double tol = 1e-10;
aztecSolver.Iterate(maxIts, tol);
```

3.3.3 Further remarks

This section is only meant to give a brief introduction on how to use MUELU as a preconditioner within the TRILINOS packages for iterative solvers. There are other, more complicated, ways to use MUELU as a preconditioner for BELOS and AZTECOO through the XPETRA interface. Of course, MUELU can also work as standalone multigrid solver. For more information on these topics, the reader may refer to the examples and tests in the MUELU source directory (`Trilinos/packages/muelu`), as well as to the MUELU tutorial [17]. For in-depth details of MUELU applied to multiphysics problems, please see [16].

Chapter 4

Performance tips

In practice, it can be very challenging to find an appropriate set of multigrid parameters for a specific problem, especially if few details are known about the underlying linear system. In this Chapter, we provide some advice for improving multigrid performance.

☛ For optimizing multigrid parameters, it is highly recommended to set the verbosity to `high` or `extreme` for MUELU to output more information (e.g., for the effect of the chosen parameters to the aggregation and coarsening process).

Some general advice:

- Choose appropriate iterative linear solver (e.g., GMRES for non-symmetric problems).
- Start with the recommended settings for particular problem types. See Table 5.1.
- Choose reasonable basic multigrid parameters (see §5.3), including maximum number of multigrid levels (`max levels`) and maximum allowed coarse size of the problem (`coarse: max size`). Take fine level problem size and sparsity pattern into account for a reasonable choice of these parameters.
- Select an appropriate transfer operator strategy (see §5.7). For symmetric problems, you should start with smoothed aggregation multigrid. For non-symmetric problems, a Petrov-Galerkin smoothed aggregation method is a good starting point, though smoothed aggregation may also perform well.
- Enable implicit restrictor construction (`transpose: use implicit`) for symmetric problems.
- Find good level smoothers (see §5.4). If a problem is symmetric positive definite, choose a smoother with a matrix-vector computational kernel, such as the Chebyshev polynomial smoother. If you are using relaxation smoothers, we recommend starting with optimizing the damping parameter. Once you have found a good damping parameter for your problem, you can increase the number of smoothing iterations.
- Adjust aggregation parameters if you experience bad coarsening ratios (see §5.5). Particularly, try adjusting the minimum (`aggregation: min agg size`) and maximum (`aggregation: max agg size`) aggregation parameters. For a 2D (3D) isotropic problem on a regular mesh, the aggregate size should be about 9 (27) nodes per aggregate.

- Replace a direct solver with an iterative method (`coarse: type`) if your coarse level solution becomes too expensive (see §5.4).

Some advice for parallel runs include:

1. Enable matrix rebalancing when running in parallel (`repartition: enable`).
2. Use smoothers invariant to the number of processors, such as polynomial smoothing, if possible.
3. Use `uncoupled` aggregation instead of `coupled`, as later requires significantly more communication.
4. Adjust rebalancing parameters (see §5.6). Try choosing rebalancing parameters so that you end up with one processor on the coarsest level for the direct solver (this avoids additional communication).
5. Enable implicit rebalancing of prolongators and restrictors (`repartition: rebalance P and R`).

Chapter 5

MueLu options

In this section, we report the complete list of MueLu input parameters. It is important to notice, however, that MueLu relies on other Trilinos packages to provide support for some of its algorithms. For instance, Ifpack/Ifpack2 provide standard smoothers like Jacobi, Gauss-Seidel or Chebyshev, while Amesos/Amesos2 provide access to direct solvers. The parameters affecting the behavior of the algorithms in those packages are simply passed by MueLu to a routine from the corresponding library. Please consult corresponding packages' documentation for a full list of supported algorithms and corresponding parameters.

5.1 Using parameters on individual levels

Some of the parameters that affect the preconditioner can in principle be different from level to level. By default, parameters affect all levels in a multigrid hierarchy.

The settings on a particular levels can be changed by using level sublists. A level sublist is a `ParameterList` sublist with the name "level XX", where XX is the level number. The parameter names in the sublist do not require any modifications. For example, the following fragment of code

```
<ParameterList name="level 2">
  <Parameter name="smoother: type" type="string" value="CHEBYSHEV"/>
</ParameterList>
```

changes the smoother for level 2 to be a polynomial smoother.

5.2 Parameter validation

By default, MueLu validates the input parameter list. A parameter that is misspelled, is unknown, or has an incorrect value type will cause an exception to be thrown and execution to halt.

☛ Spaces are important within a parameter's name. Please separate words by just one space, and make sure there are no leading or trailing spaces.

The option `print initial parameters` prints the initial list given to the interpreter. The option `print unused parameters` prints the list of unused parameters.

5.3 General options

Problem type	Multigrid algorithm	Block size	Smoother
`unknown`	–	–	–
`Poisson-2D`	Smoothed aggregation	1	Chebyshev
`Poisson-3D`	Smoothed aggregation	1	Chebyshev
`Elasticity-2D`	Smoothed aggregation	3	Chebyshev
`Elasticity-3D`	Smoothed aggregation	6	Chebyshev
`ConvectionDiffusion`	Petrov-Galerkin AMG	1	Gauss-Seidel
`MHD`	Unsmoothed aggregation	–	Additive Schwarz method with one level of overlap and ILU(0) as a subdomain solver

Table 5.1. Supported problem types ("–" means not set).

`problem: type` — [string] Type of problem to be solved. Possible values: see Table 5.1. **Default:** "unknown".

`verbosity` — [string] Control of the amount of printed information. Possible values: "none", "low", "medium", "high", "extreme". **Default:** "high".

`number of equations` — [int] Number of PDE equations at each grid node. Only constant block size is considered. **Default:** 1.

`max levels` — [int] Maximum number of levels in a hierarchy. **Default:** 10.

`cycle type` — [string] Multigrid cycle type. Possible values: "V", "W". **Default:** "V".

`problem:`	`symmetric`	[bool] Symmetry of a problem. This setting affects the construction of a restrictor. If set to true, the restrictor is set to be the transpose of a prolongator. If set to false, underlying multigrid algorithm makes the decision. **Default:** true.

5.4 Smoothing and coarse solver options

MUELU relies on other TRILINOS packages to provide level smoothers and coarse solvers. IFPACK and IFPACK2 provide standard smoothers (see Table 5.2), and AMESOS and AMESOS2 provide direct solvers (see Table 5.3). Note that it is completely possible to use any level smoother as a direct solver.

MUELU checks parameters `smoother: * type` and `coarse: type` to determine:

- what package to use (i.e., is it a smoother or a direct solver);
- (possibly) transform a smoother type

  ☛ IFPACK and IFPACK2 use different smoother type names, e.g., "point relaxation stand-alone" vs "RELAXATION". MUELU tries to follow IFPACK2 notation for smoother types. Please consult IFPACK2 documentation [8] for more information.

The parameter lists `smoother: * params` and `coarse: params` are passed directly to the corresponding package without any examination of their content. Please consult the documentation of the corresponding packages for a list of possible values.

By default, MUELU uses one sweep of symmetric Gauss-Seidel for both pre- and post-smoothing, and SuperLU for coarse system solver.

`smoother:`	`pre or post`	[string] Pre- and post-smoother combination. Possible values: "pre" (only pre-smoother), "post" (only post-smoother), "both" (both pre-and post-smoothers), "none" (no smoothing). **Default:** "both".
`smoother:`	`type`	[string] Smoother type. Possible values: see Table 5.2. **Default:** "RELAXATION".
`smoother:`	`pre type`	[string] Pre-smoother type. Possible values: see Table 5.2. **Default:** "RELAXATION".

smoother: type	
RELAXATION	Point relaxation smoothers, including Jacobi, Gauss-Seidel, symmetric Gauss-Seidel, etc. The exact smoother is chosen by specifying relaxation: type parameter in the smoother: params sublist.
CHEBYSHEV	Chebyshev polynomial smoother.
ILUT, RILUK	Local (processor-based) incomplete factorization methods.

Table 5.2. Commonly used smoothers provided by IFPACK/IFPACK2. Note that these smoothers can also be used as coarse grid solvers.

coarse: type	AMESOS	AMESOS2	
KLU	x		Default AMESOS solver [7].
KLU2		x	Default AMESOS2 solver [4].
SuperLU	x	x	Third-party serial sparse direct solver [9].
SuperLU_dist	x	x	Third-party parallel sparse direct solver [9].
Umfpack	x		Third-party solver [6].
Mumps	x		Third-party solver [2].

Table 5.3. Commonly used direct solvers provided by AMESOS/AMESOS2

smoother: post type	[string] Post-smoother type. Possible values: see Table 5.2. **Default:** "RELAXATION".
smoother: params	[ParameterList] Smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.
smoother: pre params	[ParameterList] Pre-smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.

`smoother: post params`	[ParameterList] Post-smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.
`smoother: overlap`	[int] Smoother subdomain overlap. **Default:** 0.
`smoother: pre overlap`	[int] Pre-smoother subdomain overlap. **Default:** 0.
`smoother: post overlap`	[int] Post-smoother subdomain overlap. **Default:** 0.
`coarse: max size`	[int] Maximum dimension of a coarse grid. MUELU will stop coarsening once it is achieved. **Default:** 2000.
`coarse: type`	[string] Coarse solver. Possible values: see Table **??**. **Default:** "SuperLU".
`coarse: params`	[ParameterList] Coarse solver parameters. MUELU passes them directly to the appropriate package library.
`coarse: overlap`	[int] Coarse solver subdomain overlap. **Default:** 0.

5.5 Aggregation options

`aggregation: type`	[string] Aggregation scheme. Possible values: see Table 5.4. **Default:** "uncoupled".
`aggregation: ordering`	[string] Node ordering strategy. Possible values: "natural" (local index order), "graph" (filtered graph breadth-first order), "random" (random local index order). **Default:** "natural".

`uncoupled`	Attempts to construct aggregates of optimal size (3^d nodes in d dimensions). Each process works independently, and aggregates cannot span multiple processes.
`coupled`	Attempts to construct aggregates of optimal size (3^d nodes in d dimensions). Aggregates are allowed to cross processor boundaries. **Use carefully**. If unsure, use `uncoupled` instead.

Table 5.4. Available coarsening schemes.

`aggregation:`	`drop scheme`	[string] Connectivity dropping scheme for a graph used in aggregation. Possible values: ”classical”, ”distance laplacian”. **Default:** ”classical”.
`aggregation:`	`drop tol`	[double] Connectivity dropping threshold for a graph used in aggregation. **Default:** 0.0.
`aggregation:`	`min agg size`	[int] Minimum size of an aggregate. **Default:** 2.
`aggregation:`	`max agg size`	[int] Maximum size of an aggregate (-1 means unlimited). **Default:** -1.
`aggregation:`	`Dirichlet threshold`	[double] Threshold for determining whether entries are zero during Dirichlet row detection. **Default:** 0.0.
`aggregation:`	`export visualization data`	[bool] Export data for visualization post-processing. **Default:** false.

5.6 Rebalancing options

`repartition:`	`enable`	[bool] Rebalancing on/off switch. **Default:** false.

`repartition:`	`partitioner`	[string] Partitioning package to use. Possible values: "zoltan" (ZOLTAN library), "zoltan2" (ZOLTAN2 library). **Default:** "zoltan2".
`repartition:`	`params`	[ParameterList] Partitioner parameters. MUELU passes them directly to the appropriate package library.
`repartition:`	`start level`	[int] Minimum level to run partitioner. MUELU does not rebalance levels finer than this one. **Default:** 2.
`repartition:`	`min rows per proc`	[int] Minimum number of rows per processor. If the actual number if smaller, then rebalancing occurs. **Default:** 800.
`repartition:`	`max imbalance`	[double] Maximum nonzero imbalance ratio. If the actual number is larger, the rebalancing occurs. **Default:** 1.2.
`repartition:`	`remap parts`	[bool] Postprocessing for partitioning to reduce data migration. **Default:** true.
`repartition:`	`rebalance P and R`	[bool] Explicit rebalancing of R and P during the setup. This speeds up the solve, but slows down the setup phases. **Default:** true.

5.7 Multigrid algorithm options

`multigrid algorithm`		[string] Multigrid method. Possible values: see Table 5.5. **Default:** "sa".
`semicoarsen:`	`coarsen rate`	[int] Rate at which to coarsen unknowns in the z direction. **Default:** 3.
`sa:`	`damping factor`	[double] Damping factor for smoothed aggregation. **Default:** 1.33.

`sa`	Classic smoothed aggregation [15]
`unsmoothed`	Aggregation-based, same as `sa` but without damped Jacobi prolongator improvement step
`pg`	Prolongator smoothing using A, restriction smoothing using A^T, local damping factors [12]
`emin`	Constrained minimization of energy in basis functions of grid transfer operator [18, 11]

Table 5.5. Available multigrid algorithms for generating grid transfer matrices.

`sa: use filtered matrix`	[bool] Matrix to use for smoothing the tentative prolongator. The two options are: to use the original matrix, and to use the filtered matrix with filtering based on filtered graph used for aggregation. **Default:** true.
`filtered matrix: use lumping`	[bool] Lump (add to diagonal) dropped entries during the construction of a filtered matrix. This allows user to preserve constant nullspace. **Default:** true.
`filtered matrix: reuse eigenvalue`	[bool] Skip eigenvalue calculation during the construction of a filtered matrix by reusing eigenvalue estimate from the original matrix. This allows us to skip heavy computation, but may lead to poorer convergence. **Default:** true.
`emin: iterative method`	[string] Iterative method to use for energy minimization of initial prolongator in energy-minimization. Possible values: ”cg” (conjugate gradient), ”sd” (steepest descent). **Default:** ”cg”.
`emin: num iterations`	[int] Number of iterations to minimize initial prolongator energy in energy-minimization. **Default:** 2.
`emin: num reuse iterations`	[int] Number of iterations to minimize the reused prolongator energy in energy-minimization. **Default:** 1.

`emin: pattern`	[string] Sparsity pattern to use for energy minimization. Possible values: "AkPtent". **Default:** "AkPtent".
`emin: pattern order`	[int] Matrix order for the "AkPtent" pattern. **Default:** 1.

5.8 Miscellaneous options

`export data`	[`ParameterList`] Exporting a subset of the hierarchy data in a file. Currently, the list can contain any of three parameter names ("A", "P", "R") of type `string` and value "{levels separated by commas}". A matrix with a name "X" is saved in three MatrixMarket files: a) data is saved in *X_level.mm*; b) its row map is saved in *rowmap_X_level.mm*; c) its column map is saved in *colmap_X_level.mm*.
`print initial parameters`	[bool] Print parameters provided for a hierarchy construction. **Default:** true.
`print unused parameters`	[bool] Print parameters unused during a hierarchy construction. **Default:** true.
`transpose: use implicit`	[bool] Use implicit transpose for the restriction operator. **Default:** false.

References

[1] Trilinos CMake Quickstart. http://trilinos.org/build_instructions.html, 2014.

[2] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. MUMPS: a general purpose distributed memory sparse solver. In *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pages 121–130. Springer, 2001.

[3] Roscoe A. Bartlett. Teuchos::RCP beginners guide. Technical Report SAND2004-3268, Sandia National Labs, 2010.

[4] Eric Bavier, Mark Hoemmen, Sivasankaran Rajamanickam, and Heidi Thornquist. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255, 2012.

[5] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial*. SIAM, 2nd edition, 2000.

[6] Timothy A. Davis. Algorithm 832: UMFPACK v4.3 — an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004.

[7] Timothy A. Davis and Ekanathan Palamadai Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software (TOMS)*, 37(3):36, 2010.

[8] Mark Hoemmen, Jonathan J. Hu, and Christopher Siefert. Ifpack2: incomplete factorizations, relaxations, and domain decomposition library. http://trilinos.org/packages/ifpack2, 2014.

[9] Xiaoye S. Li, James W. Demmel, John R. Gilbert, Laura Grigori, Meiyue Shao, and Ichitaro Yamazaki. SuperLU Users' Guide. 2011.

[10] Paul Lin, Matthew Bettencourt, Stefan Domino, Travis Fisher, Mark Hoemmen, Jonathan J. Hu, Eric Phipps, Andrey Prokopenko, Sivasankaran Rajamanickam, Christopher Siefert, and Stephen Kennon. Towards extreme-scale simulations for low Mach fluids with second-generation Trilinos. To appear.

[11] L. Olson, J. Schroder, and R. Tuminaro. A general interpolation strategy for algebraic multigrid using energy minimization. *SIAM Journal on Scientific Computing*, 33(2):966–991, 2011.

[12] Marzio Sala and Raymond S. Tuminaro. A new Petrov-Galerkin smoothed aggregation preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 31(1):143–166, 2008.

[13] Heidi Thornquist, Roscoe A. Bartlett, Mark Hoemmen, Christopher Baker, and Michael Heroux. Teuchos: Trilinos tools library. http://trilinos.org/packages/teuchos, 2014.

[14] Ulrich Trottenberg, Cornelis Oosterlee, and Anton Schüller. *Multigrid*. Elsevier Academic Press, 2001.

[15] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56:179–196, 1996.

[16] Tobias A. Wiesner. *Flexible Aggregation-based Algebraic Multigrid Methods for Contact and Flow Problems*. PhD thesis, 2014.

[17] Tobias A. Wiesner, Michael W. Gee, Andrey Prokopenko, and Jonathan J. Hu. The MueLu tutorial. http://trilinos.org/packages/muelu/muelu-tutorial, 2014. SAND2014-18624R.

[18] Tobias A. Wiesner, Raymond S. Tuminaro, Wolfgang A. Wall, and Michael W. Gee. Multigrid transfers for nonsymmetric systems based on Schur complements and Galerkin projections. *Numerical Linear Algebra with Applications*, 21(3):415–438, 2014.

Appendix A

Copyright and License

MueLu: A package for multigrid based preconditioning

Copyright 2012 Sandia Corporation

Under the terms of Contract DE–AC04–94AL85000 with Sandia Corporation, the U.S. Government retains certain rights in this software.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Corporation nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

DISTRIBUTION:

1 Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany

1 MS 1320 Michael Heroux, 1446
1 MS 1318 Robert Hoekstra, 1446
1 MS 1320 Mark Hoemmen, 1446
1 MS 1320 Paul Lin, 1446
1 MS 1318 Andrey Prokopenko, 1426
1 MS 1322 Christopher Siefert, 1443
1 MS 0899 Technical Library, 9536 (electronic copy)

DISTRIBUTION:

1	MS 9159	Jonathan Hu, 1426
1	MS 9159	Paul Tsuji, 1442
1	MS 9159	Raymond Tuminaro, 1442
1	MS 0899	Technical Library, 8944 (electronic copy)

Sandia National Laboratories